Adaptation dynamique des fonctionnalités d'un système d'exploitation large échelle

Djawida Dib

Adaptation dynamique des fonctionnalités d'un système d'exploitation large échelle

Djawida Dib

INRIA
Campus de Beaulieu
35042 Rennes cedex - France
djawida.dib@inria.fr

Résumé
Cet article présente notre projet de recherche qui consiste à adapter dynamiquement les fonctionnalités d'un système d'exploitation large échelle. L'objectif de cette adaptation est d'offrir la meilleure qualité de service possible aux applications qui s'exécutent sur des infrastructures distribuées.

Mots-clés : adaptation dynamique, systèmes large échelle, infrastructure distribuées, gestion de ressources, cohérence de données répliquées.

---

1. Introduction

Les dernières années ont vu l'émergence des infrastructures d'exécution résultants d'une interconnexion entre plusieurs calculateurs. Ces infrastructures sont de type grappes (*cluster*) si elles sont composées de calculateurs homogènes ou de type grille de calcul (*grid computing*) si elles sont composées de calculateurs hétérogènes. La variation de disponibilité des ressources dans ce genre d'infrastructure est permanente. D'un côté les nœuds qui les composent rejoignent et quittent dynamiquement l'infrastructure de calcul et d'un autre côté les besoins des applications qui s'exécutent sur ces infrastructures peuvent varier en permanence.
En conséquence il est très délicat de choisir une méthode de gestion de ressources qui convienne à toutes les situations. Il devient alors nécessaire d'adapter dynamiquement les fonctionnalités du système de gestion de ressources pour ces infrastructures en fonction de la disponibilité des ressources et des besoins des applications.
Cet article est organisé de la manière suivante. Nous discutons dans la section 2 les motivations de notre étude. La section 3 présente les technologies existantes qui nous serviront de support. Dans la section 4 nous montrons un premier exemple d'adaptation de la fonctionnalité de cohérence de données répliquées. Enfin la section 5 conclut cet article et expose nos perspectives.

2. Motivations

Les infrastructures de type grappe ou grille de calcul sont de nature diverse et dynamique. Cette forte dynamicité rend difficile le choix d'un algorithme pour chacune des fonctionnalités du système d'exploitation qui gère ces infrastructures. Les fonctionnalités de base d'un système d'exploitation telles que l'ordonnancement de tâches, l'allocation de ressources,... peuvent être implémentées suivant différents algorithmes. Chaque variante algorithmique est mieux adaptée à une situation donnée. La fonctionnalité de tolérance aux fautes pour les données, par exemple, peut être implémentée en dupliquant les données ou en les répartissant sur différents sites. La méthode de duplication de données garantie plus de tolérance quand il y a suffisamment de ressources interconnectées à l'infrastructure. Cependant la duplication est coûteuse en espace disque et la méthode de répartition de données sera meilleure pour

Ce projet bénéficie du soutien financier de la Région Bretagne

la globalité du système. Changer dynamiquement, suivant chaque situation, le comportement des fonctionnalités d'un système d'exploitation permet de maintenir un niveau de qualité optimal.
De leur côté les applications qui partagent une même infrastructure ont des besoins et des privilèges très différents. Il est judicieux de classer ces applications suivant certains critères (niveau de privilège des utilisateurs, type d'accès au données,...) afin d'attribuer aux fonctionnalités du système d'exploitation l'algorithme le mieux adapté à chaque classe. En reprenant l'exemple de la tolérance aux fautes pour les données, les applications peuvent choisir elles-même la méthode à utiliser pour gérer leurs données. En conséquence, plusieurs comportements d'une même fonctionnalité doivent avoir la possibilité de s'exécuter simultanément.
L'intérêt de ces adaptations est d'offrir la meilleure qualité de service possible à chaque application suivant son niveau de privilège dans le système. En contrepartie, il est nécessaire de gérer la compatibilité entre les différents algorithmes et de trouver une stratégie pour les coordonner de façon a ce qu'aucun choix d'algorithme ne pénalise un autre choix.

3. Technologies existantes

Adapter dynamiquement les fonctionnalités d'un système d'exploitation large échelle nécessite l'existence d'un tel système. Un système d'exploitation large échelle doit contenir toutes les fonctionnalités de base qui permettent l'utilisation d'une infrastructure distribuée. Une telle adaptation nécessite également un framework d'adaptation dynamique à utiliser comme support pour adapter chacune des fonctionnalités du système. En conséquence, le framework d'adaptation doit avoir la possibilité de s'exécuter au-dessus du système d'exploitation considéré et le système d'exploitation, de son côté, doit fournir des interfaces de communication pour permettre une interaction avec le framework.
Dans les sous-sections ci-dessous nous présentons respectivement les systèmes d'exploitation large échelle et le framework d'adaptation qui nous serviront pour bâtir un prototype et faire des expérimentations.

3.1. Système d'exploitation large échelle

Nous avons accès à deux systèmes d'exploitations large échelle, XtreemOS [1] pour les grilles de calcul et Kerrighed [2] pour les grappes.
Xtreemos est un système d'exploitation pour grille basé sur Linux. Il offre à une grille ce qu'un système d'exploitation traditionnel offre à un simple ordinateur. Ce système masque l'hétérogénéité des calculateurs qui le composent et partage des ressources sécurisées entre différents utilisateurs.
Kerrighed est un système d'exploitation pour grappe de type système à image unique (*Single System Image*). Il offre la vue d'une machine unique au-dessus d'une grappe de PCs standards. Kerrighed est implémenté comme une extension du système d'exploitation Linux.

3.2. Framework d'adaptation

Le framework sur lequel nous nous appuyons pour l'adaptation dynamique est nommé SAFDIS [3]. SAFDIS est un framework d'auto-adaptation des services distribués. Il permet aux applications à base de services d'évoluer dynamiquement en leur fournissant toutes les fonctionnalités du modèle MAPE [4].
Dans notre cas, SAFDIS est utilisé pour adapter dynamiquement les fonctionnalités d'un système d'exploitation large échelle. Nous considérons ces fonctionnalités comme étant des services distribués offerts aux applications.

4. Un premier exemple

Considérons un premier exemple illustrant nos motivations. Cet exemple étudie la fonctionnalité de cohérence de données répliquées. Parmi les modèles de cohérence de données répliquées existants, ceux les plus connus sont le modèle de cohérence forte et le modèle de cohérence faible.
Le modèle de cohérence forte (ou cohérence séquentielle) assure que toute lecture d'une copie d'une donnée renvoie la dernière valeur attribuée à cette donnée. Dans ce modèle il y a deux approches de synchronisation des copies répliquées [5] : l'invalidation sur écriture et la diffusion sur écriture. L'algorithme d'invalidation sur écriture, avant de modifier une donnée, invalide toutes les répliques sauf

celle à modifier. Les autres répliques, suite à un défaut de page, se chargent à la demande des processus correspondants. L'algorithme de diffusion sur écriture, avant de modifier une donnée, verrouille toutes les répliques sauf celle à modifier. Après la modification il met automatiquement à jour toutes les autres répliques puis il les déverrouille.
L'algorithme d'invalidation sur écriture est mieux adapté dans le cas de surcharge du réseau qui interconnecte les nœuds de l'infrastructure distribuée. Autrement, dans le cas ou les performances réseau sont bonnes l'algorithme de diffusion sur écriture peut offrir de meilleures performances grâce à sa réduction des défauts de page. Ainsi la sélection d'une variante algorithmique par rapport à une autre repose principalement sur les performances réseau. En pratique le délais de transmission de données entre les nœuds de l'infrastructure distribuée peut être considéré comme étant un indice de performances réseau.
Quant au modèle de cohérence faible, il assure que si une donnée est modifiée, au bout d'un certain temps toutes les copies refléteront cette modification. En d'autres termes, il n'y a aucune nécessité de mettre à jour les autres copies jusqu'à ce qu'une synchronisation prenne place, ce qui réduit largement la quantité de communication nécessaire. Ce modèle de cohérence de données est adapté aux applications l'ayant explicitement permis, comme il peut être adapté aux applications à très faible niveau de privilège.
La possibilité d'avoir ces deux modèles de cohérence de données répliquées (cohérence forte et cohérence faible) dans un même système est d'un intérêt considérable. D'une part la consommation des ressources physiques est réduite en utilisant le modèle de cohérence faible pour la classe d'applications correspondante et d'autre part la fiabilité des résultats est garantie pour le reste des applications en utilisant le modèle de cohérence forte. En l'occurrence, le modèle de cohérence faible doit avoir la possibilité de s'exécuter simultanément avec l'un des deux algorithmes du modèle de cohérence forte, soit avec l'algorithme d'invalidation sur écriture ou avec l'algorithme de diffusion sur écriture, selon les demandes des applications et les capacités du système.

5. Perspectives et conclusion

Dans cet article, nous avons discuté notre vision d'un système d'exploitation large échelle aux multiples fonctionnalités qui se compose dynamiquement suivant les besoins des applications et les architectures matérielles sous-jacentes. Nous avons également illustré notre approche par un exemple de la fonctionnalité de cohérence de données répliquées. Trois algorithmes différents ont été présentés pour l'implémentation de cette fonctionnalité. En pratique le système d'exploitation Kerrighed utilise l'algorithme d'invalidation sur écriture. Nous avons implémenté l'algorithme de diffusion sur écriture au sein de Kerrighed afin d'effectuer une première expérimentation.
Notre plan de travail, pour faire aboutir cette étude, est de commencer d'abord par approfondir les critères de sélection d'algorithme et étudier ensuite la façon et le coût de transition entre les algorithmes. La cohabitation de différentes variantes algorithmiques sera étudiée par la suite. Nous comptons également étudier d'autres fonctionnalités telles que l'ordonnancement de tâches, la tolérance aux fautes,... et mettre en évidence les paramètres influençant leur comportement. La coordination entre les algorithmes de différentes fonctions sera également étudier afin de finaliser notre approche.
Notre objectif est d'offrir la meilleure qualité de service possible aux applications en permettant une interaction entre la couche applicative et la couche système. Cette interaction nécessite l'ajout d'une interface évoluée au-dessus de la couche système. Le bénéfice d'avoir une telle interface est de permettre d'une part aux applications de préciser au système d'exploitation la manière de les gérer, d'autre part au système d'exploitation d'informer la couche applicative de l'état des ressources physiques et des algorithmes qui peuvent les gérer. Ceci a pour but de faire profiter de toutes les adaptations possibles aux deux niveaux.

Bibliographie

1. http ://www.xtreemos.eu/
2. http ://www.kerrighed.org/
3. Guillaume Gauvrit et al. – SAFDIS : A Framework to Bring Self-Adaptability to Service-Based Distributed

Applications. – 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA) (2010) 211–218

4. J. O. Kephart et D. M. Chess – The vision of autonomic computing. – Computer, vol. 36, no. 1, 2003.
5. Kai Li – Memory coherence in shared virtual memory systems. – ACM Transactions on Computer Systems, Volume 7 Issue 4, Nov. 1989 practice. – Prentice Hall, 1988.